Web Information Technology Project

Academic Year 2023-2024

Track 1: Image Gallery - RIA Version

Luca De Nicola, Filippo Raimondi
Personal Code: 10808901
Personal Code: 10809051

Contents

1 Database Design

1.1 ER Diagram

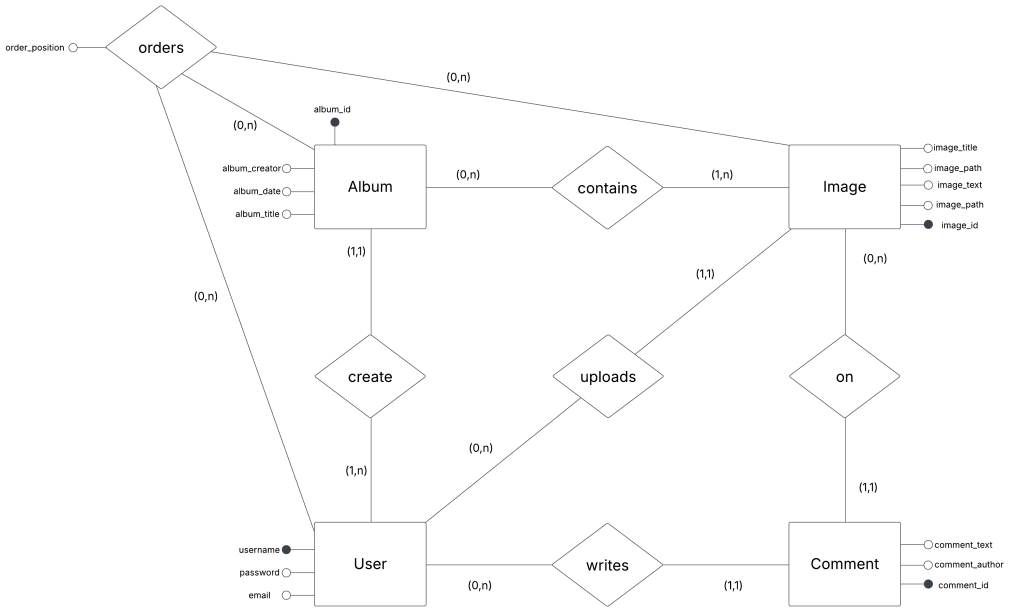


Figure 1: ER Diagram of the Database

1.2 Database Schema

```sql
CREATE TABLE 'User' (
  'username' varchar(32) NOT NULL,
  'email' varchar(64) NOT NULL,
  'password' varchar(128) NOT NULL,
  PRIMARY KEY ('username'),
  UNIQUE KEY 'email_UNIQUE' ('email'),
  UNIQUE KEY 'username_UNIQUE' ('username')
);
```

```sql
CREATE TABLE 'Album' (
  'album_id' int NOT NULL AUTO_INCREMENT,
  'album_creator' varchar(32) NOT NULL,
  'album_title' varchar(64) NOT NULL,
  'album_date' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY ('album_id'),
  KEY 'album_creator_idx' ('album_creator'),
  CONSTRAINT 'album_creator' FOREIGN KEY ('album_creator')
    REFERENCES 'User' ('username')
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE 'AlbumContainsImage' (
  'album_id' int NOT NULL,
  'image_id' int NOT NULL,
  PRIMARY KEY ('album_id','image_id'),
  KEY 'id_image_idx' ('image_id'),
  CONSTRAINT 'id_album' FOREIGN KEY ('album_id') REFERENCES '
      Album' ('album_id') ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT 'id_image' FOREIGN KEY ('image_id') REFERENCES '
      Image' ('image_id') ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE 'Image' (
  'image_id' int NOT NULL AUTO_INCREMENT,
  'image_uploader' varchar(32) NOT NULL,
  'image_title' varchar(64) NOT NULL,
  'image_date' timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  'image_text' varchar(512) NOT NULL,
  'image_path' varchar(256) NOT NULL,
  PRIMARY KEY ('image_id'),
  UNIQUE KEY 'image_path_UNIQUE' ('image_path'),
  KEY 'image_uploader_idx' ('image_uploader'),
  CONSTRAINT 'image_uploader' FOREIGN KEY ('image_uploader')
      REFERENCES 'User' ('username') ON DELETE CASCADE ON UPDATE
      CASCADE
);
```

```
CREATE TABLE 'Comment' (
  'comment_id' int NOT NULL AUTO_INCREMENT,
  'image_id' int NOT NULL,
  'comment_author' varchar(32) NOT NULL,
  'comment_text' varchar(512) NOT NULL,
  PRIMARY KEY ('comment_id'),
  KEY 'image_id_idx' ('image_id'),
  KEY 'comment_author_idx' ('comment_author'),
  CONSTRAINT 'comment_author' FOREIGN KEY ('comment_author')
      REFERENCES 'User' ('username') ON DELETE CASCADE ON UPDATE
      CASCADE,
  CONSTRAINT 'image_id' FOREIGN KEY ('image_id') REFERENCES '
      Image' ('image_id') ON DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE 'UserImageOrder' (
    'username' varchar(32) NOT NULL,
    'album_id' int NOT NULL,
    'image_id' int NOT NULL,
    'order_position' int NOT NULL,
    PRIMARY KEY ('username','album_id','image_id'),
    KEY 'album_id_idx' ('album_id'),
    KEY 'image_id_idx' ('image_id'),
    CONSTRAINT 'fk_userimageorder_album_id' FOREIGN KEY ('
        album_id') REFERENCES 'Album' ('album_id') ON DELETE
        CASCADE ON UPDATE CASCADE,
    CONSTRAINT 'fk_userimageorder_image_id' FOREIGN KEY ('
        image_id') REFERENCES 'Image' ('image_id') ON DELETE
        CASCADE ON UPDATE CASCADE,
    CONSTRAINT 'fk_userimageorder_username' FOREIGN KEY ('
        username') REFERENCES 'User' ('username') ON DELETE
        CASCADE ON UPDATE CASCADE
);
```

2 Components

2.1 Server Components

2.1.1 Model

- Album
- Comment
- Image
- User

2.1.2 Utils

The utils folder contains a set of support classes that simplify the development and maintenance of the application without directly belonging to a single layer (Model, DAO, Controller, or View). Specifically

- **DatabaseConnectionPool**: Provides a connection pooling system for database connections, optimizing performance and managing the creation and release of connections.
- **DatabaseListener**: Initializes and destroys the connection pool when the application starts or shuts down (through the contextInitialized and contextDestroyed methods).
- **PasswordEncrypt**: Provides methods for password hashing and verification.
- **StringUtil**: Offers functions for string validation and manipulation (e.g., checking emails, usernames, passwords, texts, etc.), centralizing control logic.
- **MimeDetector**: This class provides methods to identify the MIME type of images (JPEG, PNG, WEBP) by analyzing the byte signatures of the data. Additionally, it allows retrieving the file extension corresponding to the detected MIME type.

2.1.3 DAOs

- **UserDAO**
  - `registerUser(user)` : boolean
  - `loginUser(user)` : boolean
  - `isUsernameTaken(username)` : boolean
  - `isEmailTaken(email)` : boolean
  - `getUsernameByEmail(email)` : String
  - `deleteUser(username)` : boolean

- **AlbumDAO**
  - `getMyAlbums(username)` : List of Album
  - `getOtherAlbums(username)` : List of Album
  - `createAlbum(album)` : boolean
  - `getAlbumsCountByUser(username)` : int
  - `getUserPersonalAlbumId(username)` : int
  - `isAlbumOwnedByUser(albumId, username)` : boolean
  - `getMyAlbumIds(username)` : List of int
  - `getAlbumById(albumId)` : Album
  - `doesAlbumExist(albumId)` : boolean
  - `getImagesByAlbumId(albumId)` : List of Image
  - `getImagesCountByAlbumId(albumId)` : int
  - `getImagesByAlbumIdWithPagination(albumId, pageSize, startIndex)` : List of Image
  - `getImagesWithCommentsByAlbumId(albumId)` : `Map<Image, List<Comment>>`
- **CommentDAO**
  - `getCommentsCountByUser(username)` : int
  - `getCommentsByImageId(imageId)` : List of Comment
  - `addComment(comment)` : boolean
- **ImageDAO**
  - `getImagesCountByUser(username)` : int
  - `addImage(image)` : int
  - `updateImagePath(imageId, imagePath)` : boolean
  - `addImageToAlbums(imageId, albumIds)` : boolean
  - `doesImageExist(imageId)` : boolean
  - `doesImageBelongToAlbum(imageId, albumId)` : boolean
  - `getImageById(imageId)` : Image
  - `doesImageBelongToUser(imageId, username)` : boolean
  - `deleteImageById(imageId)` : boolean
  - `getImagePathById(imageId)` : String
- **UserImageOrderDAO**
  - `userHasImagesOrderForAlbum(username, albumId)` : boolean
  - `deleteUserImagesOrderForAlbum(username, albumId)` : boolean
  - `saveUserImagesOrderForAlbum(username, albumId, imageIds)` : boolean
  - `getUserImagesOrderForAlbum(username, albumId)` : List of Integer

2.1.4 Controllers

- AlbumServlet
- ErrorServlet
- HomeServlet
- ImageServlet
- IndexServlet
- UploadsServlet
- SpaServlet

2.2 Client Components

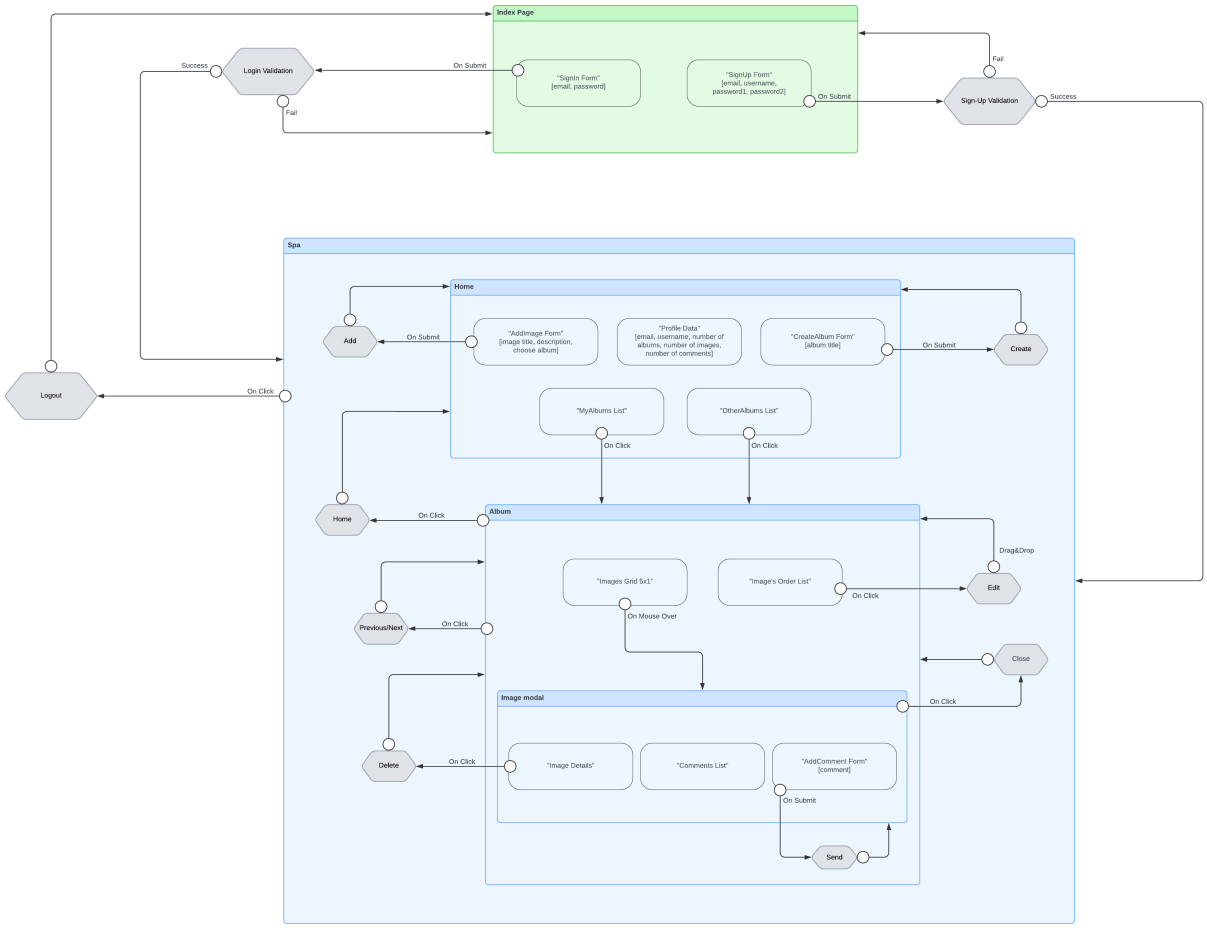
2.2.1 Views

- error.html
- index.html
- spa.html

2.2.2 Js

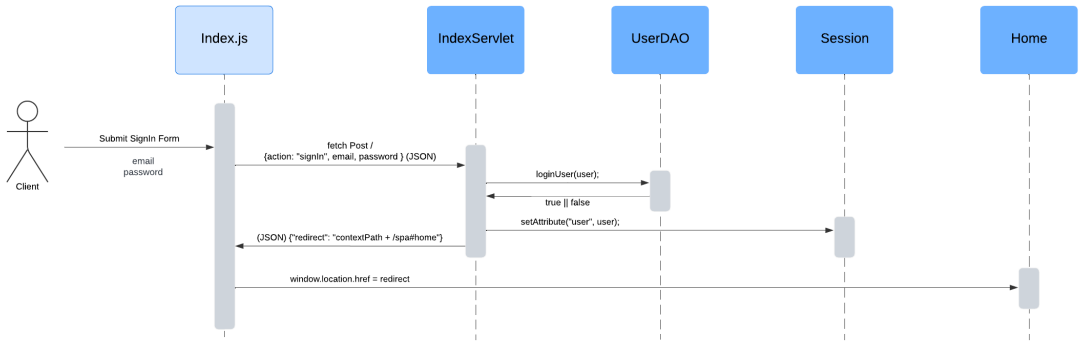
- spa.js
- album.js
- home.js
- image.js
- index.js
- StringUtil.js

3 IFML Diagram

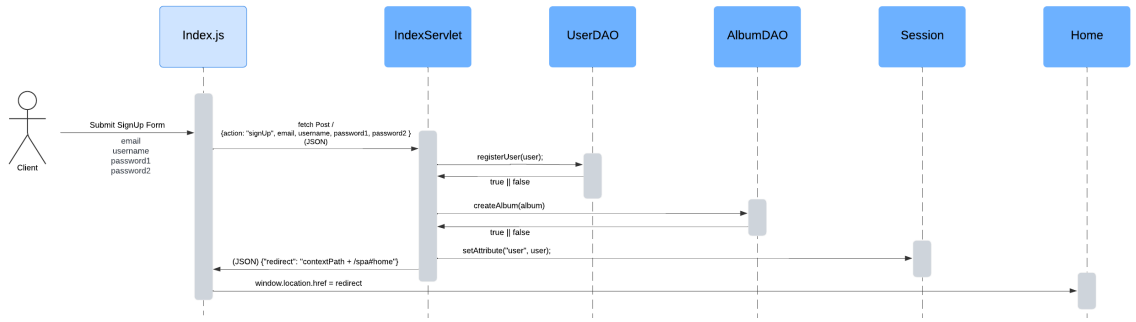


4 Sequence Diagram

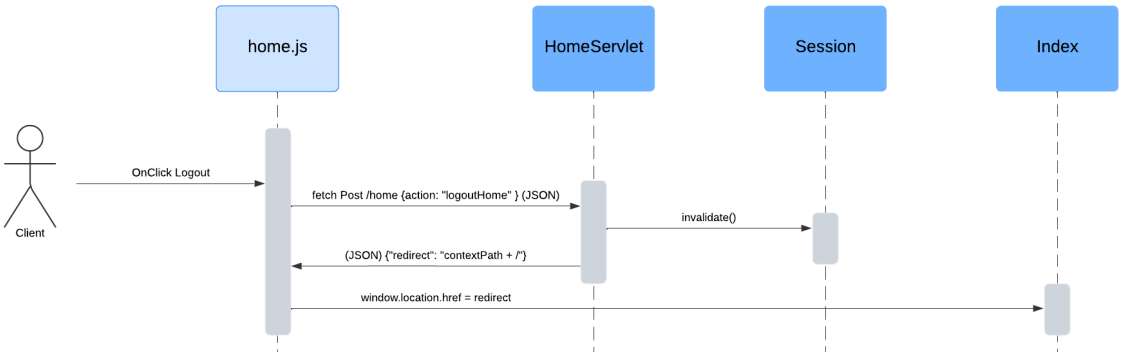
4.1 Event: Sign In



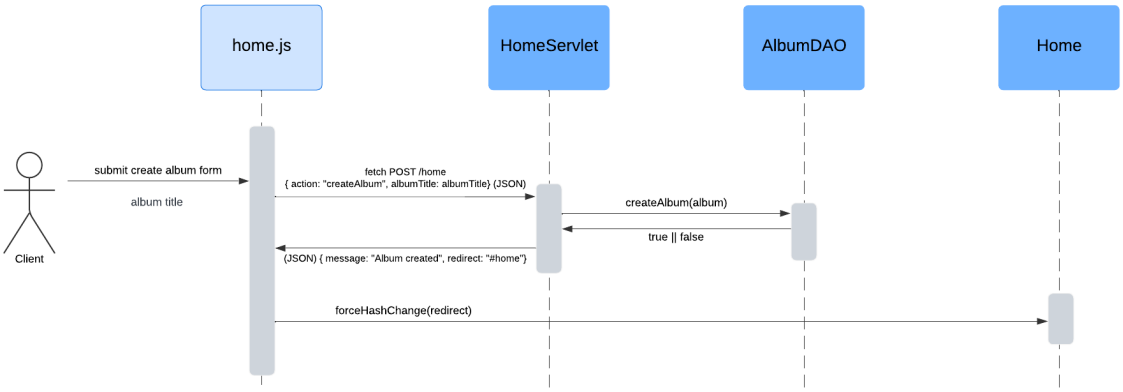
4.2 Event: Sign Up



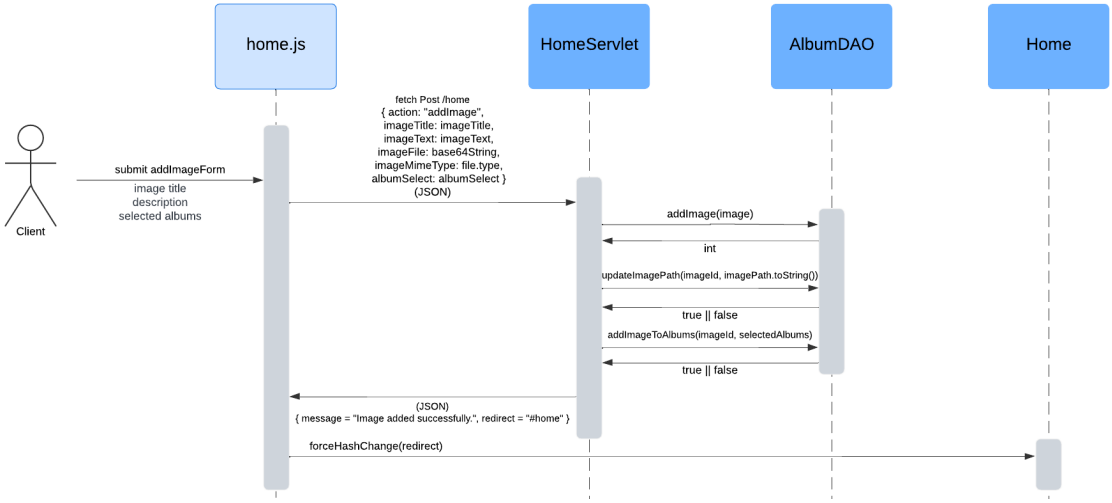
4.3 Event: Logout



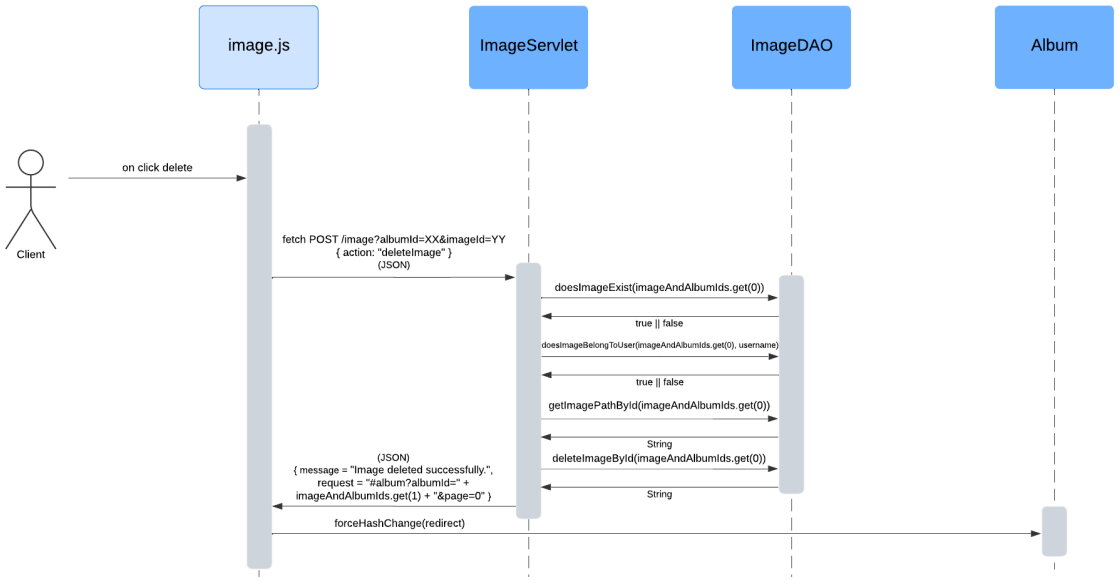
4.4 Event: Create Album



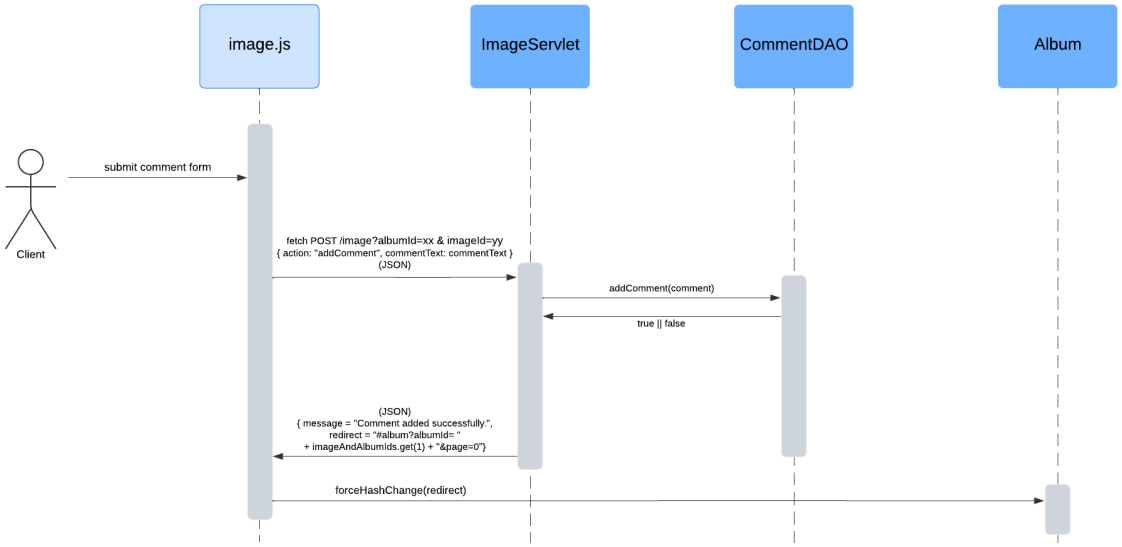
4.5 Event: Add Image



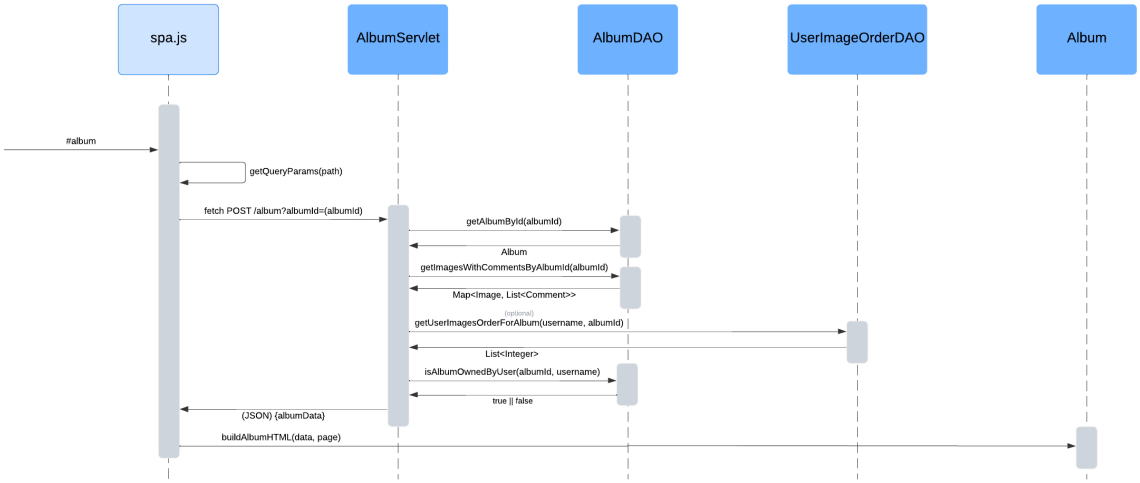
4.6 Event: Delete Image



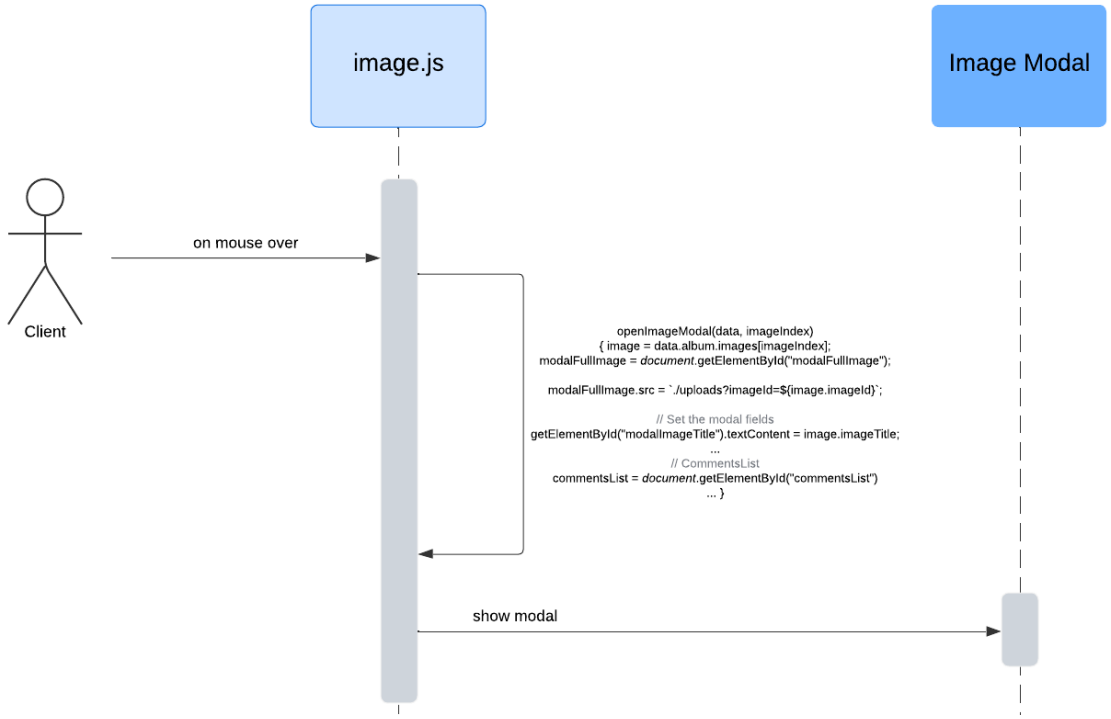
4.7 Event: Add Comment



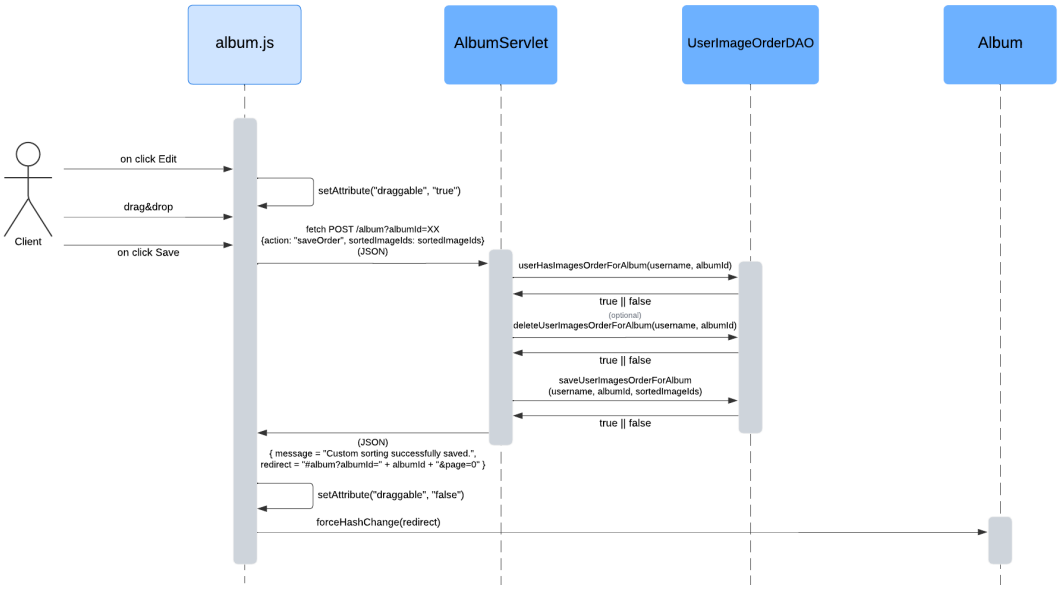
4.8 Event: Render Album



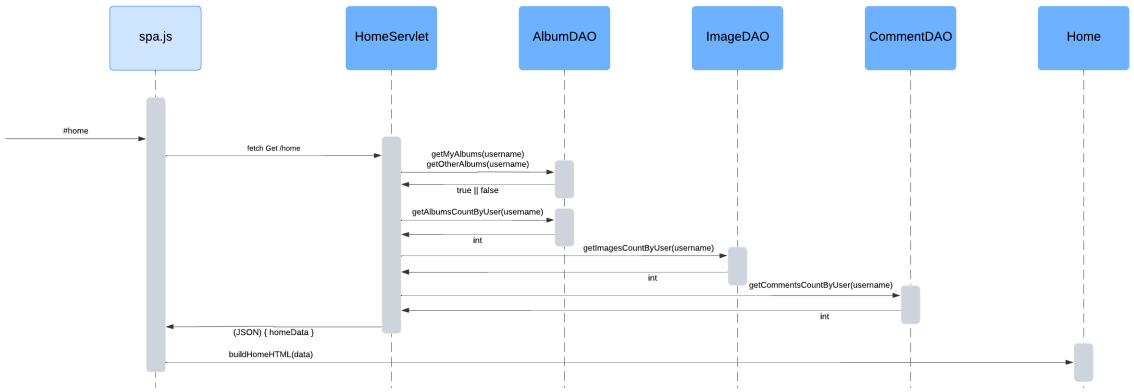
4.9 Event: View Image (Modal)



4.10 Event: Reorder Images (DragDrop)



4.11 Event: Render Home



5 Actions and Events

Client Side: Event	Client Side: Action
1) loginForm → submit (Sign In)	JS validates fields, then `fetch POST /?action=signIn {email, password}`
2) signUpForm → submit	JS validates fields, then `fetch POST /?action=signUp {email, username, ...}`
3) Homepage → load (#home)	JS detects `#home` and calls `GET /home`
4) createAlbumForm → submit	JS checks `albumTitle`, then `fetch POST /home {action=createAlbum, ...}`
5) addImageForm → submit	JS reads file → Base64, then `fetch POST /home {action=addImage, imageFile=base64}`
6) Album Page → load (#album?albumId=XX)	JS reads `#album?albumId=XX`, calls `GET /album?albumId=XX`
7) (Reorder) Save Order → click	JS serializes `sortedImageIds`, then `fetch POST /album?albumId=XX {action=saveOrder, ...}`
8) addCommentForm → submit	JS checks `commentText`, then `fetch POST /image?albumId=..., imageId=... {action=addComment, ...}`
9) Open Image Modal	JS retrieves `imageData` from `data.album.images[]`, populates `<div id="modal">`
10) Delete Image form → submit	JS calls `fetch POST /image?albumId=..., imageId=... {action=deleteImage}`
11) Logout (from Home or Album)	JS: `fetch POST /home {action=logoutHome}` or `/album {action=logoutAlbum}`

Server Side: Event	Server Side: Action
1) IndexServlet doPost(signIn)	Validates credentials, sets session, responds in JSON with `redirect="/spa#home"`
2) IndexServlet doPost(signUp)	Checks email/username, creates user (`UserDAO`), creates `@username` album, sets session → `redirect="/spa#home"`
3) HomeServlet doGet	Retrieves `myAlbums, otherAlbums, userStats`, responds in JSON
4) HomeServlet doPost(createAlbum)	Validates, calls `AlbumDAO.createAlbum`, responds JSON success/error (`redirect="#home"`)
5) HomeServlet doPost(addImage)	Decodes Base64, calls `ImageDAO.addImage`, saves on disk, responds JSON success/error
6) AlbumServlet doGet	Loads album info + images + comments, responds JSON
7) AlbumServlet doPost(saveOrder)	Saves the order in `UserImageOrderDAO`, responds JSON success/error
8) ImageServlet doPost(addComment)	Calls `CommentDAO.addComment`, responds JSON success/error, with `redirect="#album?..."`
9) No server event	Data already loaded in JSON; the "modal" logic is purely client-side
10) ImageServlet doPost(deleteImage)	Checks ownership in DB, deletes record + file, responds in JSON
11) HomeServlet / AlbumServlet doPost(logout...)	`session.invalidate()`, JSON `redirect="/"`